🎵 Strudel: Programar Música desde Cero

Tutorial en Español para Principiantes Absolutos

> **¿Para quién es este tutorial?** Para ti, que nunca has programado ni sabes leer música. No necesitas ningún conocimiento previo. Solo necesitas curiosidad, un navegador web y ganas de explorar.

📌 Cómo usar este tutorial

Este documento tiene **tres partes principales**:

1. **Capítulo 0** — Contexto: ¿qué es programar y de dónde viene Strudel?
2. **Capítulo 1** — Tu primer sonido: aprender haciendo
3. **Capítulo 2** — Ritmos, secuencias y patrones musicales

En cada sección encontrarás:

- 📖 Explicación clara del concepto
- 💻 Código listo para copiar y pegar
- 🎯 Ejercicios para practicar
- 💡 Tips y notas importantes

🌐 Antes de empezar: abre el entorno

Strudel funciona directamente en tu navegador. No necesitas instalar nada.

1. Abre esta dirección en tu navegador: **https://strudel.cc/**
2. Verás un editor de texto con código de colores
3. Borra todo lo que hay en el editor (Ctrl+A y luego Suprimir)
4. ¡Listo! Ya puedes empezar a escribir tu primer código

Comandos esenciales que usarás todo el tiempo:

Acción	Teclas
▶ Ejecutar / actualizar código	`Ctrl + Enter`
■ Detener todos los sonidos	`Ctrl + .`

CAPÍTULO 0

¿Qué es programar y de dónde viene Strudel?

0.1 ¿Qué es un lenguaje de programación?

Imagina que necesitas pedirle a alguien que haga algo muy específico, pero esa persona solo entiende instrucciones escritas en un idioma completamente exacto, donde cada palabra, cada coma y cada punto tiene un significado preciso. Eso es, en esencia, un **lenguaje de programación**.

Un lenguaje de programación es un conjunto de reglas (una *sintaxis*) que permite a los seres humanos darle instrucciones a una computadora. La computadora es muy poderosa pero también muy literal: hace exactamente lo que le dices, nada más y nada menos.

Una analogía cotidiana

Piensa en una receta de cocina. La receta es un *programa*:

- Tiene una lista de ingredientes (los *datos*)
- Tiene una secuencia de pasos (las *instrucciones*)
- Si sigues la receta correctamente, obtienes el plato (el *resultado*)

Un lenguaje de programación funciona igual, pero en lugar de cocinar, le dices a la computadora qué calcular, qué mostrar en pantalla... o en nuestro caso, **qué sonidos producir**.

¿Por qué existen tantos lenguajes?

Existen cientos de lenguajes de programación porque cada uno fue diseñado para resolver problemas diferentes. Es como los idiomas humanos: el español es ideal para ciertas expresiones poéticas, el inglés para los negocios globales, el mandarín para una cultura milenaria. Del mismo modo:

- **Python** es ideal para analizar datos científicos

- **JavaScript** es el lenguaje de las páginas web interactivas
- **C++** se usa para videojuegos y sistemas que necesitan mucha velocidad
- Y lenguajes como **Strudel** o **TidalCycles** fueron diseñados especialmente para **crear música en tiempo real**

0.2 ¿Qué es el Live Coding?

El *live coding* (código en vivo) es una práctica artística y musical que nació a finales de los años 1990 en universidades del Reino Unido. La idea es radical y hermosa: **el código se escribe y se modifica mientras la música está sonando**, frente a una audiencia.

Imagina a un DJ o a un músico de jazz improvisando. El live coder hace algo similar, pero en lugar de tocar un instrumento, **escribe y reescribe código** para transformar la música en tiempo real. El error también es parte del espectáculo: si algo falla, la música cambia inesperadamente, y eso también puede ser arte.

Esta práctica está vinculada con movimientos de **arte digital, software libre y cultura hacker**, donde se valora la transparencia del proceso creativo. Los proyectores muestran el código al público, rompiendo la ilusión de "magia tecnológica" y diciendo: *aquí está todo, tú también puedes hacer esto.*

0.3 De TidalCycles a Strudel: una genealogía

Para entender Strudel, hay que conocer su origen.

TidalCycles (2009)

Alex McLean, artista e investigador de la Universidad de Leeds (Reino Unido), creó **TidalCycles** en 2009. McLean quería un lenguaje que tratara la música como *patrones matemáticos*, de forma similar a cómo los matemáticos piensan en funciones y transformaciones.

TidalCycles está escrito en **Haskell**, un lenguaje de programación muy abstracto y elegante (pero también difícil de instalar). Para usarlo, necesitas instalar varios programas complejos en tu computador, lo que crea una barrera de entrada alta.

La filosofía de TidalCycles es profunda: la música no son notas aisladas, sino **patrones que se transforman en el tiempo**. Un ritmo de batería es un patrón. Una melodía es un patrón. Incluso el silencio es un patrón.

Strudel (2022)

Felix Roos, desarrollador y músico alemán, tomó la filosofía y el lenguaje de TidalCycles y lo tradujo a **JavaScript** — el lenguaje de los navegadores web. El resultado es **Strudel**: un sistema de

live coding que funciona directamente en el navegador, sin instalar absolutamente nada.

El nombre "Strudel" (el pastelito en espiral) es un guiño al nombre "Tidal" (que en inglés alude al movimiento cíclico del mar), y también hace referencia a los **patrones en espiral y cíclicos** que son fundamentales en la música y en este lenguaje.

> 💡 **Para reflexionar:** Tanto TidalCycles como Strudel nacieron en el contexto académico europeo, pero hoy los usa gente en todo el mundo — desde artistas de Ciudad de México hasta músicos de Lagos o Bangkok. ¿Qué dice eso sobre cómo viaja el conocimiento hoy?

0.4 ¿Cómo piensa Strudel la música?

Strudel tiene una idea muy particular de qué es la música: **todo es un patrón cíclico**.

En lugar de pensar en una canción como una línea de tiempo que va del inicio al fin (como una grabación), Strudel piensa en la música como **ciclos que se repiten y se transforman**. Piénsalo como un reloj de arena que se voltea infinitamente, o como los tambores en un ritual que nunca tienen un "principio" ni un "final" absolutos.

Cada ciclo en Strudel dura **2 segundos por defecto**, pero eso se puede cambiar. Dentro de cada ciclo, Strudel distribuye los eventos musicales que le indiques.

Esta visión cíclica de la música conecta con tradiciones musicales de África, Asia y América Latina, donde el **ritmo cíclico** (no la melodía lineal de la tradición europea) es el centro de la experiencia musical.

CAPÍTULO 1

Tus Primeros Sonidos

1.1 La función `sound()`: reproducir un sonido

Llegó el momento de hacer sonar algo. La función más básica de Strudel es `sound()`.

¿Qué es una función?

Una **función** es como un comando que le das a Strudel. Le dices "haz esto" y Strudel lo hace. Las funciones en Strudel siempre van seguidas de paréntesis `()`, y dentro de esos paréntesis le pasas **información** (llamada *argumento*).

Tu primer código

Escribe esto en el editor de Strudel y presiona `Ctrl + Enter`:

```javascript
sound("casio")
```

¡Deberías escuchar un sonido de teclado casio que se repite!

Si no escuchas nada, puede ser que el navegador pida permiso de audio. Busca un aviso en la barra del navegador y permite el audio.

¿Qué significa esto?

- `sound` → es la función que le dice a Strudel "reproduce un sonido"
- `"casio"` → es el nombre del sonido que quieres usar (va entre comillas)

Las **comillas** `" "` son muy importantes: le dicen a Strudel "esto es un texto, no un comando". Si escribes `sound(casio)` sin comillas, Strudel se confundirá y habrá un error.

Probemos otros sonidos

Reemplaza `casio` con cada uno de estos nombres (uno a la vez) y escucha cómo suenan:

```javascript
sound("insect")
```

```javascript
sound("wind")
```

```javascript
sound("jazz")
```

```javascript
sound("metal")
```

```javascript
sound("east")
```

```javascript
```

```
sound("crow")
```

```
javascript
sound("space")
```

🎯 **Ejercicio 1:** Escucha todos los sonidos de la lista y elige tus tres favoritos. Anótalos, los usarás más adelante.

1.2 Variantes de un sonido con `:número`

Muchos sonidos en Strudel no son solo uno, sino que son **familias de sonidos similares**. Por ejemplo, `jazz` no es solo un sonido de jazz: hay varios instrumentos o tomas grabadas del mismo tipo.

Para seleccionar una variante específica, usas `:` seguido de un número:

```
javascript
sound("jazz:0")
```

```
javascript
sound("jazz:1")
```

```
javascript
sound("jazz:2")
```

```
javascript
sound("jazz:3")
```

> 💡 **Nota:** Si no escribes número (como en `sound("jazz")`), es lo mismo que `sound("jazz:0")`. Los números empiezan siempre en **0**, no en 1. En programación esto es muy común y se llama "indexación desde cero".

🎯 **Ejercicio 2:** Explora las variantes del sonido `crow` y del sonido `numbers`. ¿Cuántas variantes diferentes puedes encontrar antes de que se repitan o dejen de sonar diferente?

1.3 Sonidos de batería: el lenguaje del ritmo

Strudel tiene una colección de sonidos de batería con nombres abreviados. Estos nombres vienen de la tradición del *drum machine* (máquina de ritmos) en la música electrónica:

```javascript
sound("bd hh sd oh")
```

¡Acabas de crear un ritmo básico de batería! Cada palabra es un instrumento diferente:

Código	Inglés	En español
`bd`	bass drum	bombo
`sd`	snare drum	caja / redoblante
`hh`	hi-hat	charles (cerrado)
`oh`	open hi-hat	charles (abierto)
`lt`	low tom	tom bajo
`mt`	mid tom	tom medio
`ht`	high tom	tom agudo
`rd`	ride cymbal	platillo ride
`cr`	crash cymbal	platillo crash
`rim`	rimshot	aro de la caja
`cp`	clap	palmada / clap

> 🥁 **Para entender mejor:** Imagina una batería física. El `bd` es el tambor grande del suelo que se toca con el pie. El `sd` es el tambor que suena como un golpe seco y brillante. Los `hh` son los dos platillos pequeños que están uno encima del otro.

🎯 **Ejercicio 3:** Crea tu propio patrón de batería usando al menos 4 instrumentos diferentes.

1.4 La función `bank()`: cambiando la máquina de ritmos

La misma nota de batería puede sonar muy diferente dependiendo de la **máquina de ritmos** que

la produce. En los años 70, 80 y 90, diferentes marcas fabricaron máquinas que tienen sonidos icónicos:

```javascript
sound("bd hh sd oh").bank("RolandTR909")
```

Cambia `RolandTR909` por cada uno de estos nombres y escucha las diferencias:

```javascript
sound("bd hh sd oh").bank("AkaiLinn")
```

```javascript
sound("bd hh sd oh").bank("RhythmAce")
```

```javascript
sound("bd hh sd oh").bank("RolandTR808")
```

```javascript
sound("bd hh sd oh").bank("RolandTR707")
```

```javascript
sound("bd hh sd oh").bank("ViscoSpaceDrum")
```

¿Qué es ese punto `.` entre `sound(...)` y `.bank(...)`?

Ese punto se llama **encadenamiento** o *method chaining*. Significa "aplica esta función adicional al resultado anterior". Es como agregar instrucciones: primero reproduce este sonido, *y además* usa este banco de sonidos.

Más adelante usaremos muchos de estos encadenamientos para agregar efectos, cambiar el volumen, el tempo, etc.

> 🎵 **Dato cultural:** La **Roland TR-808** (1980) es posiblemente la máquina de ritmos más influyente de la historia. Sus sonidos de bombo profundo y caja aguda definieron el hip-hop, el house, el techno, el reggaeton y decenas de géneros más. ¡Escucha con atención cómo suena el `bd` con ese banco!

🎯 **Ejercicio 4:** Compara cómo suena exactamente el mismo patrón `"bd sd bd sd"` con `RolandTR808` vs `RolandTR909`. Describe en dos palabras la diferencia de cada uno.

CAPÍTULO 2

Secuencias, Ritmos y Patrones

> Este es el corazón de Strudel. Todo lo que aprendas aquí te permitirá construir composiciones increíblemente complejas.

2.1 Secuencias: varios sonidos en orden

Cuando escribes varios nombres dentro de `sound("...")` separados por espacios, Strudel los reproduce **en secuencia**, uno después del otro, dentro de un ciclo:

```javascript
sound("bd hh sd hh")
```

Esto crea un ritmo básico que se repite: bombo, charles, caja, charles.

La regla del ciclo

Recuerda: Strudel divide cada ciclo (2 segundos por defecto) en partes iguales según cuántos elementos tenga la secuencia.

- `sound("bd sd")` → 2 elementos, cada uno dura 1 segundo
- `sound("bd hh sd hh")` → 4 elementos, cada uno dura 0.5 segundos
- `sound("bd hh sd hh oh bd")` → 6 elementos, cada uno dura ~0.33 segundos

Esto significa que **mientras más elementos agregas, más rápido suena**. Pruébalo:

```javascript
sound("bd bd hh bd rim bd hh bd")
```

🎯 **Ejercicio 5:** Escribe una secuencia con 2 elementos y luego otra con 8. ¿Puedes escuchar claramente la diferencia en velocidad?

2.2 El silencio: `-` y `~`

El silencio es tan importante como el sonido en la música. En Strudel, usas `-` o `~` para indicar un tiempo de silencio:

```javascript
sound("bd hh - rim - bd hh rim")
```

Aquí, las posiciones con `-` son silencios. El patrón suena: bombo, charles, *silencio*, aro, *silencio*, bombo, charles, aro.

> 💡 **Tanto `-` como `~` funcionan igual** para el silencio. Puedes usar el que prefieras o mezclarlos.

🎯 **Ejercicio 6:** Toma el patrón `"bd sd bd sd"` y agrega silencios para crear un ritmo sincopado (con acentos en lugares inesperados).

2.3 Ángulos `< >`: un elemento por ciclo

Los corchetes angulares `< >` cambian cómo Strudel trata una secuencia: en lugar de reproducir todos los elementos en un ciclo, reproduce **solo uno por ciclo**, alternando entre ellos:

```javascript
sound("<bd bd hh bd rim bd hh bd>")
```

Esto suena lento porque en cada ciclo de 2 segundos solo escuchas un golpe. Para hacerlo más interesante:

```javascript
sound("<bd bd hh bd rim bd hh bd>*8")
```

El `*8` significa "repite esto 8 veces dentro del ciclo". Ahora suena rítmico de nuevo, pero con una propiedad especial: **el tempo no cambia** si agregas o quitas elementos.

¿Por qué es útil `< >`?

Imagina que tienes una secuencia de acordes que quieres que cambien lentamente, uno por ciclo. Los `< >` son perfectos para eso. Más adelante los usaremos mucho con notas y acordes.

2.4 El tempo: `setcpm()`

`setcpm` significa "set cycles per minute" (fijar ciclos por minuto). Controla la velocidad de todo lo que estás creando:

```javascript
setcpm(90/4)
sound("<bd hh rim hh>*8")
```

¿Cómo calcular el tempo?

La fórmula es: `setcpm(BPM/4)`

- 60 BPM (muy lento) → `setcpm(60/4)` → `setcpm(15)`
- 90 BPM (moderado) → `setcpm(90/4)` → `setcpm(22.5)`
- 120 BPM (dance music estándar) → `setcpm(120/4)` → `setcpm(30)`
- 140 BPM (techno rápido) → `setcpm(140/4)` → `setcpm(35)`

> 🎵 **BPM** = Beats Per Minute = Pulsaciones por minuto. Es la medida estándar de tempo en música. El corazón humano en reposo late a unos 60-70 BPM. La mayoría de la música pop está entre 100-130 BPM.

🎯 **Ejercicio 7:** Toma el siguiente patrón y prueba 3 tempos diferentes (lento, medio, rápido):

```javascript
setcpm(120/4)
sound("bd*4, - cp, hh*8")
```

2.5 Sub-secuencias: `[ ]` brackets

Los corchetes `[ ]` te permiten meter una secuencia *dentro* de otra. El contenido del corchete ocupa el mismo espacio que si fuera un solo elemento:

```javascript
sound("bd [hh hh] sd [hh bd] bd - [hh sd] cp")
```

Desglosemos esto:

- `bd` → un bombo (ocupa 1 espacio)
- `[hh hh]` → dos charles que se comprimen en 1 espacio (suenan más rápido)
- `sd` → una caja

- `[hh bd]` → charles y bombo comprimidos en 1 espacio
- etc.

Sub-sub-secuencias: `[[ ]]`

¡Puedes anidar corchetes todo lo que quieras!

```javascript
sound("bd [[rim rim] hh] bd cp")
```

Aquí, `[[rim rim] hh]` es una sub-secuencia que contiene otra sub-secuencia. Los dos `rim` se comprimen en la mitad del espacio de `[rim hh]`, que a su vez se comprime en 1 espacio.

> 🧩 **Analogía:** Imagina las muñecas rusas *matryoshka*. Cada muñeca puede contener una más pequeña dentro. Los corchetes en Strudel funcionan igual: secuencias dentro de secuencias.

🎯 **Ejercicio 8:** Crea un patrón de batería que use al menos dos sub-secuencias con `[ ]`. Intenta lograr que suene como un redoble de caja.

2.6 Multiplicación `*`: acelerar elementos

El operador `*` multiplica la velocidad de un elemento o una sub-secuencia:

```javascript
sound("bd hh*2 rim hh*3 bd [- hh*2] rim hh*2")
```

- `hh*2` → el charles se repite 2 veces en el espacio de 1
- `hh*3` → el charles se repite 3 veces en ese espacio
- `[- hh*2]` → silencio y luego dos charles, todo en el espacio de 1

También funciona con sub-secuencias completas:

```javascript
sound("bd [hh rim]*2 bd [hh rim]*1.5")
```

Y puede llegar a velocidades extremas:

```javascript
```

```
sound("bd hh*32 rim hh*16")
```

> 🎧 **¿Qué pasa cuando algo va muy rápido?** Cuando un ritmo es suficientemente rápido, el cerebro ya no lo percibe como ritmo sino como **tono musical**. Esto es la relación entre ritmo y melodía: son el mismo fenómeno a diferentes velocidades. Los científicos llaman a este umbral aproximadamente 20 Hz (20 eventos por segundo).

2.7 Paralelismo `,`: varias líneas a la vez

La coma `,` te permite tocar **múltiples secuencias al mismo tiempo**, como diferentes instrumentos de una banda:

```javascript
sound("hh hh hh, bd casio")
```

Esto toca simultáneamente:

- Línea 1: `hh hh hh` (tres charles)
- Línea 2: `bd casio` (bombo y casio)

¡Puedes tener tantas líneas como quieras!

```javascript
sound("hh hh hh, bd bd, - casio")
```

Las comas también funcionan dentro de sub-secuencias:

```javascript
sound("hh hh hh, bd [bd,casio]")
```

Aquí, `[bd,casio]` significa "bombo y casio *al mismo tiempo*" en ese espacio.

2.8 Múltiples líneas con `` ` ` `` (backticks)

Cuando tienes muchas capas, puedes usar **backticks** `` ` ` `` en lugar de comillas para escribir en múltiples líneas (es más fácil de leer):

```javascript
```

```
sound(`bd*2, - cp,
- - - oh, hh*4,
[- casio]*2`)
```

> 💡 El backtick `` ` `` está en la tecla a la izquierda del `1` en los teclados en inglés. En teclados en español puede estar en otra ubicación (a veces junto a la tecla de la ñ o arriba de la tecla tab).

2.9 La función `n()`: seleccionar muestras de forma separada

En lugar de escribir `jazz:0 jazz:1 jazz:2`, puedes separar el nombre del sonido y el número:

Forma larga:

```javascript
sound("jazz:0 jazz:1 jazz:4 jazz:2 jazz:3")
```

Forma corta con `n()`:

```javascript
n("0 1 4 2 3").sound("jazz")
```

Ambas hacen exactamente lo mismo, pero la segunda es más limpia y fácil de leer, especialmente cuando hay muchos números.

> 💡 Nota cómo aquí `n()` va **antes** del punto y `sound()` va **después**. El orden importa: `n()` genera una secuencia de números y `sound()` le dice qué sonido usar con esos números.

🎯 **Ejercicio 9:** Recrea este patrón usando `n()`:

```javascript
sound("crow:0 crow:2 crow:4 crow:1 crow:3")
```

2.10 Patrones de ejemplo: de lo simple a lo complejo

Ahora que conoces los elementos básicos, veamos cómo se construyen patrones musicales reales.

Beat de rock básico

```javascript
setcpm(100/4)
sound("[bd sd]*2, hh*8").bank("RolandTR505")
```

Esto es el patrón clásico de rock: bombo y caja alternando (`[bd sd]*2` = dos veces dentro del ciclo), con el charles golpeando 8 veces por ciclo (en corcheas).

House clásico

```javascript
sound("bd*4, [- cp]*2, [- hh]*4").bank("RolandTR909")
```

El house tiene el bombo en cada beat (`bd*4`), el clap en los beats 2 y 4 (`[- cp]*2` = silencio-clap, silencio-clap), y el charles entre los beats.

We Will Rock You (Queen)

```javascript
setcpm(81/2)
sound("bd*2 cp").bank("RolandTR707")
```

¡El ritmo más famoso del rock simplificado a dos líneas de código!

Simulación de un secuenciador de 16 pasos

```javascript
setcpm(90/4)
sound(`
[-  -  oh -] [-  -  -  -] [-  -  -  -] [-  -  -  -],
[hh hh -  -] [hh -  hh -] [hh -  hh -] [hh -  hh -],
[-  -  -  -] [cp -  -  -] [-  -  -  -] [cp -  -  -],
[bd -  -  -] [-  -  -  bd] [-  -  bd -] [-  -  -  bd]
`)
```

Este último ejemplo imita cómo funcionan las máquinas de ritmos físicas: una cuadrícula de 16 pasos donde cada celda puede estar activada o desactivada.

📋 Resumen de Funciones y Mini-Notación

Mini-Notación (el lenguaje del ritmo)

Concepto	Símbolo	Ejemplo	¿Qué hace?
Secuencia	espacio	`"bd sd hh"`	Reproduce en orden
Silencio	`-` o `~`	`"bd - sd -"`	Pausa en ese momento
Alternancia	`< >`	`"<bd sd>"`	Uno por ciclo
Sub-secuencia	`[ ]`	`"bd [hh hh]"`	Comprime en 1 espacio
Aceleración	`*n`	`"hh*4"`	Repite n veces
Paralelismo	`,`	`"bd*4, hh*8"`	Suena al mismo tiempo
Variante	`:n`	`"jazz:2"`	Muestra n del sonido

Funciones principales

Función	¿Qué hace?	Ejemplo
`sound("nombre")`	Reproduce un sonido	`sound("bd sd")`
`.bank("nombre")`	Cambia el banco de sonidos	`.bank("RolandTR808")`
`setcpm(número)`	Fija el tempo	`setcpm(120/4)`
`n("números")`	Selecciona variantes	`n("0 1 2").sound("jazz")`

🚀 Próximos pasos

Con lo que aprendiste aquí ya puedes crear ritmos y patrones de percusión. El siguiente nivel te espera:

- **Notas musicales:** aprenderás a tocar melodías con `note()`
- **Escalas y acordes:** descubrirás cómo Strudel maneja la armonía
- **Efectos de audio:** `reverb`, `delay`, `filter` y más
- **Síntesis:** crear sonidos desde cero con osciladores

Visita **https://strudel.cc/workshop/first-notes/** para continuar el tutorial oficial en inglés, o espera el próximo capítulo de este tutorial en español.

🎯 Proyecto Final del Capítulo

Crea una composición usando todos los elementos aprendidos. Tu pieza debe tener:

1. Al menos **3 líneas** simultáneas (usando `,`)
2. Al menos **una sub-secuencia** (usando `[ ]`)
3. Al menos **una aceleración** (usando `*`)
4. Un **tempo definido** con `setcpm()`
5. Un **banco de sonidos** con `.bank()`

Aquí un punto de partida que puedes modificar:

```javascript
setcpm(110/4)
sound(`
bd*4,
[- cp]*2,
hh*8,
[- - oh -]*2
`).bank("RolandTR808")
```

¡Modifícalo y hazlo tuyo!

Tutorial creado para estudiantes de pregrado. Basado en la documentación oficial de Strudel (strudel.cc) y los tutoriales de Strudel de Shovon Saha.

Strudel fue creado por Felix Roos y es un port de TidalCycles de Alex McLean.